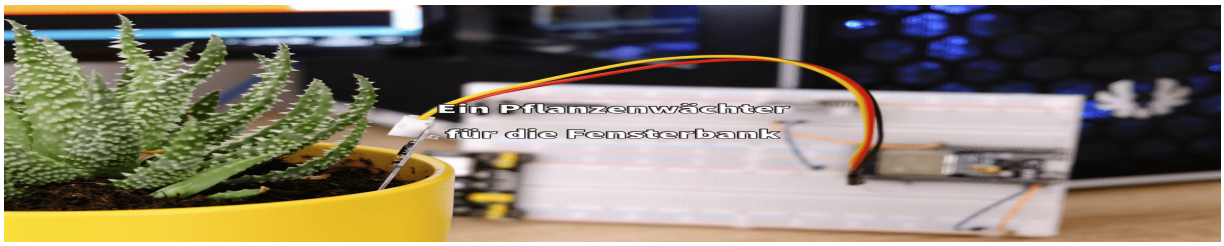


Pflanzenwächter für die Fensterbank Teil 4 – Temperatur und Luftfeuchtigkeitsmessung

Hallo und Willkommen zu einem weiteren Teil der Pflanzenwächterreihe.

Heute werden wir dem Pflanzenwächter einen 2 in 1 Umweltsensor spendieren, sowie einige Optimierungen der (LED-)Hardware als auch des Codes implementieren. Der Sensor, den wir in diesem Teil anschließen, ist der bekannte DHT 22 Temperatur und Luftfeuchtigkeitssensor. Damit kann unser Pflanzenwächter zukünftig neben der Bodenfeuchte jetzt auch die für Pflanzen in Ihrer Umgebung wichtigen Umwelt-Parameter Temperatur und Luftfeuchtigkeit erfassen. Natürlich werden wir auch unsere Handy APP passend dazu erweitern. Zuallererst nehmen wir eine Änderung der Beschaltung der LED's vor. Wir verwenden dazu das KY-016 LED RGB Modul und schließen dieses direkt wie folgt an:

Rot an Pin 0 , Grün an Pin 15, und Blau an Pin . Vorteile dieses Moduls sind, das wir zum einen bei Einbau in ein Gehäuse Platz sparen (1/3 des Gehäuseplatzes) , aber auch per PWM Ansteuerung der einzelnen Kanäle über das volle RGB Spektrum an Farben verfügen, sodass darüber mehr als ursprünglich nur 3 Zustände angezeigt werden können.

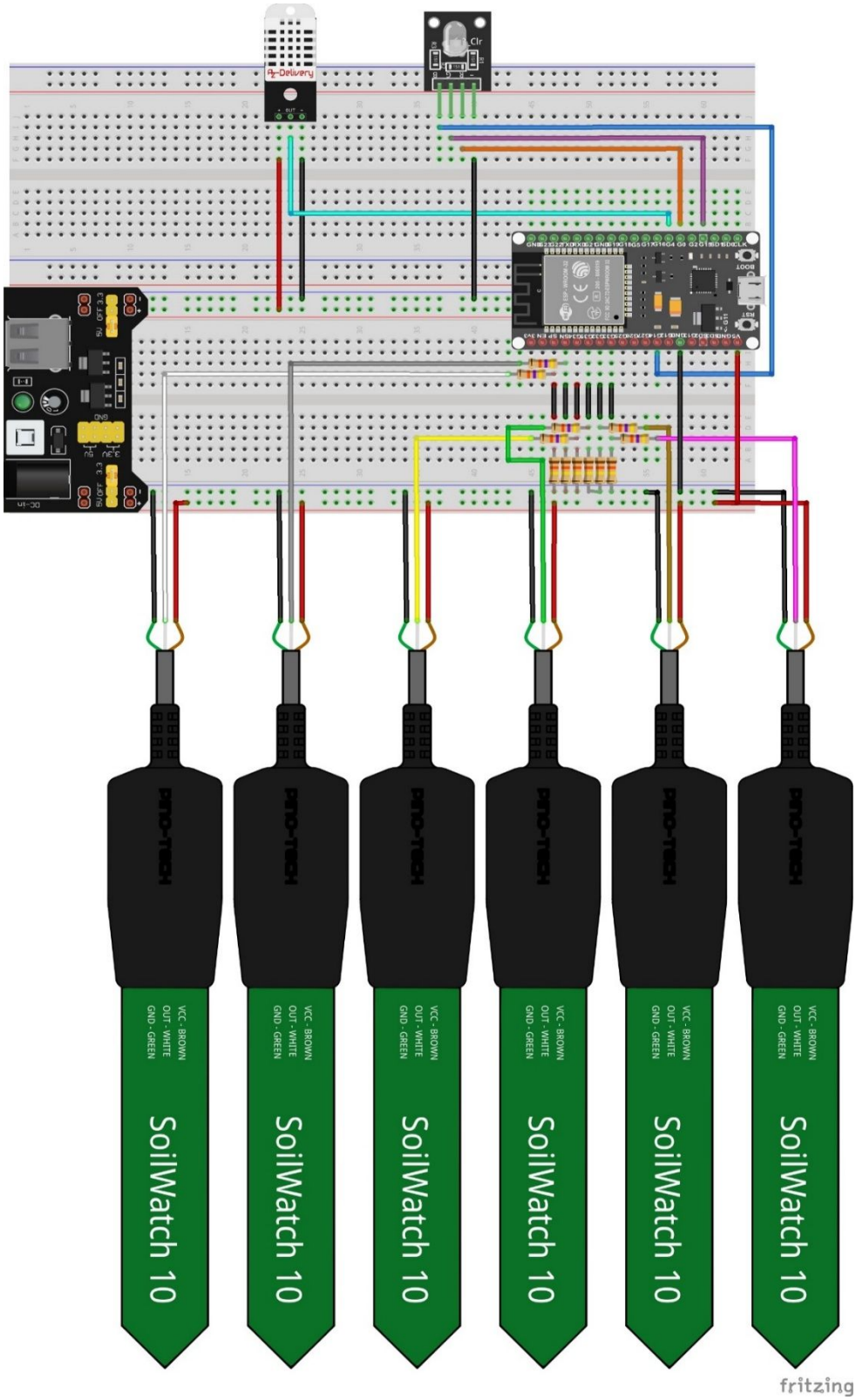
(Beispielsweise leuchtet die LED in der Initialisierungsphase jetzt in weiß). Nicht zuletzt sparen wir uns die Vorwiederstände ein, da diese jetzt bereits auf dem Modul integriert sind. Die Pins, an denen die RGB LED's an den ESP32 angeschlossen sind, wurden nach dem Kriterium "für Timing unkritische Hardware geeignet" ausgewählt. Hintergrund ist, dass die Pins...

an den die LED anschlossen ist, Firmware bedingt während eines Systemneustarts schwingen (PWM) bevor sie initialisiert werden, und somit für Timing kritische Hardware (wie den DHT22) nicht geeignet sind.

Eine Änderung hat sich auch in der Funktionsweise und der Darstellung der LED Anzeige ergeben: Da wir nur eine und nicht sechs RGB LED‘s haben um Statis des Wächters anzuzeigen, müssen wir also die wichtigsten Informationen darauf zusammenfassen.

Dazu greifen wir auf folgende Überlegung zurück: Für uns ist es die wichtigste Information, ob eine der maximal 6 Pflanzen gegossen werden sollte oder nicht. D.h sollte auf einem Blick dies erkennbar sein. Dazu definieren wir die Farbe grün (Ausreichend Feuchtigkeit) so um, das ALLE 6 Pflanzen genügend Feuchtigkeit haben, damit wir eine „grüne“ Anzeige bekommen. Sollte also einer der sechs

Sensoren weniger als 50% Bodenfeuchte detektiere, springt unsere Anzeige auf „gelb“. Bei unter 10% auf „rot“. Somit wissen wir also das wir unsere Pflanzenwächter APP starten sollten, um zu erfahren, welche der sechs Pflanzen gegossen werden sollte. Da der Temperatur und Feuchtesensor DHT 22 3,3 V Pin und Logikkompatibel ist, schließen wir ihn direkt an den Port 4 des ESP's an. Hier sehr ihr die komplette Beschaltung:



Die aktuelle Teileliste mit den benötigten Hardwareteilen. (Mit Bezugslink)

Anzahl	Beschreibung	Anmerkung
1	DHT 22	
	DHT 11	Alternativ zu DHT 22
1	KY-016 LED RGB Modul	
1	ESP-32 Dev Kit C	
6	Bodenfeuchte Sensor Modul V1.2	
1	MB102 Netzteil Adapter	Für Breadboardaufbau
12	Wiederstände laut Beschreibung	

Anstelle des zuerst gelisteten DHT 22 Sensors auch der preisgünstigere DHT 11 Sensor durch einfaches Anpassen der Zeile „DHTTYPE DHT22“verwendet werden. Der DHT 11ist jedoch nicht so Messgenau wie der DHT 22.

Hier finden sich die Unterschiede der beiden Sensoren auch noch mal zum Nachlesen.

Wenn der DHT 11 Senor verwendet werden soll , muss die Zeile

```
#define DHTTYPE DHT22
```

in

```
#define DHTTYPE DHT11
```

geändert werden. Weitere Änderungen sind nicht notwendig.

Die Pinbelegung es ESP32 ist wie folgt

ESP32 PIN	Verwendung	Anmerkung
4	DHT Sensor Eingang	
0	RGB LED PIN	Rot
15	RGB LED PIN	Grün
14	RGB LED PIN	Blau

Dies waren auch schon die notwendigen Hardwareänderungen. Um den DHT22/DHT 11 Sensor mit seinem proprietärem one Wire Protocol in unserem Sketch nutzen zu können, müssen wir als nächstes zwei weitere Bibliotheken in unsere IDE einbinden.

Dies ist zum einen die gerneralisierte „Adafruit Unified Sensor Libary“ : als auch die eigentliche DHT Sensor Library. Beide Bibliotheken bauen aufeinander auf und müssen zu unseren Bibliotheken der IDE hinzugefügt werden, da sie von unserem Projekt benötigt werden.

Nach Hinzufügen der Bibliotheken und Anpassung der Parameter im Code , wie im Teil 3 dieser Reihe, beschrieben laden wir folgenden Code auf unseren ESP hoch:

```
#include <driver/adc.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "EEPROM.h"
#include <Preferences.h>
#include "DHT.h"    // REQUIRES the following Arduino libraries:
                    //- DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
                    //- Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor

// Portedefinierung RGP LED Modul
#define LED_Rot     0     // Rote LED
#define LED_Blau    14    // Blaue LED
#define LED_Gruen   15    // Gruene LED

// LED PWM Einstellungen
#define PWMfreq 5000  // 5 Khz Basisfrequenz für LED Anzeige
#define PWMledChannelA  0
#define PWMledChannelB  1
#define PWMledChannelC  2
#define PWMresolution  8 // 8 Bit Resolution für LED PWM

//Sonstige Definitionen
#define ADCAttenuation ADC_ATTEN_DB_11    //ADC_ATTEN_DB_11 = 0-3,6V  Dämpfung ADC (ADC Erweiterung
#define MoisureSens_Poll_Interval 300000   // Intervall zwischen zwei Bodenfeuchtemessungen in Millisekunden  -> alle 5 Minuten Datenpaket an Handy senden
#define DHT_Poll_Interval 400000          // Intervall zwischen zwei Temeperatur und Luftfreuchtemessungen in Millisekunden -> alle 6 Minuten Datenpaket an Handy senden
#define MaxSensors 6                      // Maximale Anzahl an anschließbaren FeuchteSensoren
#define StartInit true
#define RunTime false
#define Sens_Calib true
#define Sens_NOTCalib false
#define EEPROM_SIZE 512                   // Größe des Internen EEPROMS

// Blynk APP Definitionen
#define BLYNK_GREEN     "#23C48E"
#define BLYNK_BLUE      "#04C0F8"
#define BLYNK_YELLOW    "#ED9D00"
#define BLYNK_RED       "#D3435C"
#define BLYNK_BLACK     "#000000"
#define BLYNK_PRINT Serial
#define BLYNK_NO_BUILTIN
#define BLYNK_NO_FLOAT
//#define BLYNK_DEBUG

//DHT Konfiguration
#define DHTPIN 4     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

struct MoistureSensorCalibrationData
 {
   int Data[MaxSensors*2] = {1651,2840,1652,2840,1653,2840,1654,2840,1655,2840,1656,2840};   // Calibration Data für Feuchtigkeitssensor. Bitte Projekt Text beachten und Werte ensprechend anpassen
  //  int Data[MaxSensors*2] = {0,0,0,0,0,0,0,0,0,0,0,0,}; // Calibration Data für Feuchtigkeitssensor. Bitte Projekt Text beachten und Werte ensprechend anpassen
   byte StatusBorderPercentValues[MaxSensors*2][2]= { {10,50},           // Zweidimensinonales Array für Prozentgrenzwerte (Ampel) jeweils Einzeln pro Feuchtesensor (1 -6)
                                        {10,50},
                                        {10,50},
                                        {10,50},
                                        {10,50},
                                        {10,50}};
  String SensorName[MaxSensors+2] = {"Pflanze 1","Pflanze 2","Pflanze 3","Pflanze 4","Pflanze 5","Pflanze 6","Luftfeuchte","Temperatur"}; // Sensorname, der auch in der APP als Überschrift angezeigt wird

 };

struct MoistureSensorData
 {
   int Percent[MaxSensors] = {0,0,0,0,0,0};       // Feuchtigkeitssensordaten in Prozent
```

```
   byte Old_Percent[MaxSensors] = {0,0,0,0,0,0};   // Vorherige _ Feuchtigkeitssensordaten in Prozent (Zweck: DatenMenge einsparen.)
   bool DataValid [MaxSensors] = {false,false,false,false,false,false};
  };

struct DHTSensorData
  {
    float Humidity = 0 ;      // Luftfeuchtigkeitssensordaten in Prozent
    float Temperature = 0;
    float Old_Humidity = 0 ;      // Luftfeuchtigkeitssensordaten in Prozent
    float Old_Temperature = 0;
    bool DataValid  = false;
    bool SensorEnabled  = false;
  };

DHT dht(DHTPIN, DHTTYPE); // DHP Instalz initalisieren


//Global Variables


char auth[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; // Hier lt. Anleitung Auth Token dere Blynk App eintragen (E-Mail).
// Deine WiFi Zugangsdaten.
char ssid[] = "Deine_WLAN_SSID";                // Bitte an eigene WLAN SSID anpassen
char pass[] = "Dein _WLAN _Passwort!";          // Bitte an eigene WLAN Passwort anpassen
MoistureSensorCalibrationData MCalib;
MoistureSensorData MMeasure;
DHTSensorData  DHTMeasure;

byte AttachedMoistureSensors; // Detected Active Moisture Sensors (Count)
unsigned long Moisure_ServiceCall_Handler = 0;  // Delay Variable for Delay between Moisure Readings
unsigned long DHT_ServiceCall_Handler = 0;  // Delay Variable for Delay between DHT Readings


void setup() {
  pinMode(LED_Rot,OUTPUT);
  pinMode(LED_Blau,OUTPUT);
  pinMode(LED_Gruen,OUTPUT);
  Serial.begin(115200);   // initialize serial communication at 115200 bits per second:
  ledcSetup(PWMledChannelA, PWMfreq, PWMresolution);
  ledcSetup(PWMledChannelB, PWMfreq, PWMresolution);
  ledcSetup(PWMledChannelC, PWMfreq, PWMresolution);
  ledcAttachPin(LED_Rot, PWMledChannelA);   // attach the channel to the GPIO to be controlled
  ledcAttachPin(LED_Blau, PWMledChannelB);
  ledcAttachPin(LED_Gruen, PWMledChannelC);
  SetLedConfig(255,255,255);
  Serial.println(F("Systemkonfiguration:"));
   if (!EEPROM.begin(EEPROM_SIZE))
  {
    Serial.println(F("failed to initialise EEPROM"));
  } else
  { Serial.println(EEPROM_SIZE);
    Serial.println(F(" Bytes EEPROM"));
  }

  AttachedMoistureSensors = DetectMoistureSensors();
  Serial.print(AttachedMoistureSensors);
  Serial.println(F(" Bodenfeuchtigkeitsensor(en)"));
  dht.begin();
  DHTMeasure.SensorEnabled  = Run_DHTSensor (StartInit);
  if (DHTMeasure.SensorEnabled)
    {
    Serial.println(F("1 DHT 22 Sensor"));
    }
  Serial.print(F("Verbindung zu WLAN"));
  delay(500);
  Blynk.begin(auth, ssid, pass);  // Initalize WiFi-Connection over Blync Library
  Serial.println(F(" erfolgreich."));
  SetLedConfig(0,0,0);
  Init_Blynk_APP();
  Run_MoistureSensors(StartInit);
  for (int i = AttachedMoistureSensors;i < 6;i++) { Update_Blynk_APP(i,Sens_Calib); };
}

byte DetectMoistureSensors ()
  {
  #define MinSensorValue 100
```

```
  byte Detected = 0;
  for (int i = 0;i < MaxSensors;i++)
    {
    int MSensorRawValue = ReadMoistureSensor_Raw_Val(i);
    if ( MSensorRawValue > MinSensorValue) { Detected++; } else {break;}
    }
  if (Detected < 1)
    {
      Serial.println(F("Keine Bodenfeuchtigkeitssesoren erkannt. System angehalten."));
      esp_deep_sleep_start();
      while(1) {}
    }
  return Detected;
  }

bool SetLedConfig(byte Red,byte Green,byte Blue)
{
ledcWrite(PWMledChannelA, Red); // Rote LED
ledcWrite(PWMledChannelB, Blue); // Blaue LED
ledcWrite(PWMledChannelC, Green); // Gruene LED
return true;
}

int ReadMoistureSensor_Raw_Val(byte Sensor)
  {
   int ReturnValue,i;
   long sum = 0;
   #define NUM_READS 6
   adc1_config_width(ADC_WIDTH_BIT_12);   //Range 0-4095
   switch (Sensor)
    {
    case 0:
      {
      adc1_config_channel_atten(ADC1_CHANNEL_0,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_0 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    case 1:
      {
      adc1_config_channel_atten(ADC1_CHANNEL_3,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_3 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    case 2:
      {
      adc1_config_channel_atten(ADC1_CHANNEL_6,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_6 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    case 3:
      {
      adc1_config_channel_atten(ADC1_CHANNEL_7,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_7 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    case 4:
      {
      adc1_config_channel_atten(ADC1_CHANNEL_4,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_4 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    default:
```

```
      {
      adc1_config_channel_atten(ADC1_CHANNEL_5,ADCAttenuation);
      for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
      sum += adc1_get_raw( ADC1_CHANNEL_5 ); //Read analog
      }
      ReturnValue = sum / NUM_READS;
      break;
      }
    }

  return ReturnValue;
  }

void Init_Blynk_APP()
  {
    Blynk.setProperty(V1,"label",MCalib.SensorName[0]);
    Blynk.setProperty(V2,"label",MCalib.SensorName[1]);
    Blynk.setProperty(V3,"label",MCalib.SensorName[2]);
    Blynk.setProperty(V4,"label",MCalib.SensorName[3]);
    Blynk.setProperty(V5,"label",MCalib.SensorName[4]);
    Blynk.setProperty(V6,"label",MCalib.SensorName[5]);
    Blynk.setProperty(V7,"label",MCalib.SensorName[6]);
    Blynk.setProperty(V8,"label",MCalib.SensorName[7]);
  }

void Update_Local_Display()
{
  byte red1 = 0;
  byte yellow1 = 0;
  byte green1 = 0;
  for (byte i = 0;i < AttachedMoistureSensors;i++)
    {
    if (MMeasure.DataValid[i])
       {
       if ( MMeasure.Percent[i] > MCalib.StatusBorderPercentValues[i][1])
          {
          green1++;
          } else if ( MMeasure.Percent[i] > MCalib.StatusBorderPercentValues[i][0])
          {
          yellow1++;
          } else
          {
          red1++;
          }
       }
    }
  if (red1 > 0)
    { SetLedConfig(255,0,0); }
  else if (yellow1 > 0)
    { SetLedConfig(255,255,0); }
  else if (green1 > 0)
    { SetLedConfig(0,255,0); }
  else
    { SetLedConfig(0,0,255); }
}

void Update_Blynk_APP(byte Sensor,bool Calibrated)
  {
    switch (Sensor)
    {
    case 0:
      {
      if ((MMeasure.DataValid[0]) & (Calibrated))
        {
        if ( MMeasure.Percent[0] > MCalib.StatusBorderPercentValues[0][1])
           {
           Blynk.setProperty(V1,"color",BLYNK_GREEN);
           } else if ( MMeasure.Percent[0] > MCalib.StatusBorderPercentValues[0][0])
           {
           Blynk.setProperty(V1,"color",BLYNK_YELLOW);
           } else
           {
           Blynk.setProperty(V1,"color",BLYNK_RED);
           }
        delay(100);
        Blynk.virtualWrite(V1,MMeasure.Percent[0]);
        } else
```

```
      {
      if (Calibrated)
        {
        Blynk.setProperty(V1,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V1,0);
        delay(100);
        Blynk.setProperty(V1,"color",BLYNK_BLACK);
        }
      else
        {
        Blynk.virtualWrite(V1,0);
        delay(100);
        Blynk.setProperty(V1,"color",BLYNK_BLUE);
        }
      }
    break;
    }
  case 1:
    {
    if ((MMeasure.DataValid[1]) & (Calibrated))
      {
      if ( MMeasure.Percent[1] > MCalib.StatusBorderPercentValues[1][1])
          {
          Blynk.setProperty(V2,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[1] > MCalib.StatusBorderPercentValues[1][0])
          {
          Blynk.setProperty(V2,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V2,"color",BLYNK_RED);
          }
      delay(100);
      Blynk.virtualWrite(V2,MMeasure.Percent[1]);
      } else
      {
      if (Calibrated)
        {
        Blynk.setProperty(V2,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V2,0);
        delay(100);
        Blynk.setProperty(V2,"color",BLYNK_BLACK);
        }
      else
        {
        Blynk.virtualWrite(V2,0);
        delay(100);
        Blynk.setProperty(V3,"color",BLYNK_BLUE);
        }
      }
    break;
    }
  case 2:
    {
    if ((MMeasure.DataValid[2]) & (Calibrated))
      {
      if ( MMeasure.Percent[2] > MCalib.StatusBorderPercentValues[2][1])
          {
          Blynk.setProperty(V3,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[2] > MCalib.StatusBorderPercentValues[2][0])
          {
          Blynk.setProperty(V3,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V3,"color",BLYNK_RED);
          }
      delay(100);
      Blynk.virtualWrite(V3,MMeasure.Percent[2]);
      } else
      {
      if (Calibrated)
        {
        Blynk.setProperty(V3,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V3,0);
        delay(100);
```

```
        Blynk.setProperty(V3,"color",BLYNK_BLACK);
        }
      else
        {
        Blynk.virtualWrite(V3,0);
        delay(100);
        Blynk.setProperty(V3,"color",BLYNK_BLUE);
        }
      }
    break;
    }
  case 3:
    {
    if ((MMeasure.DataValid[3]) & (Calibrated))
      {
      if ( MMeasure.Percent[3] > MCalib.StatusBorderPercentValues[3][1])
          {
          Blynk.setProperty(V4,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[3] > MCalib.StatusBorderPercentValues[3][0])
          {
          Blynk.setProperty(V4,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V4,"color",BLYNK_RED);
          }
      delay(100);
      Blynk.virtualWrite(V4,MMeasure.Percent[3]);
      } else
      {
      if (Calibrated)
        {
        Blynk.setProperty(V4,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V4,0);
        delay(100);
        Blynk.setProperty(V4,"color",BLYNK_BLACK);
        }
      else
        {
        Blynk.virtualWrite(V4,0);
        delay(100);
        Blynk.setProperty(V4,"color",BLYNK_BLUE);
        }
      }
    break;
    }
  case 4:
    {
    if ((MMeasure.DataValid[4]) & (Calibrated))
      {
      if ( MMeasure.Percent[4] > MCalib.StatusBorderPercentValues[4][1])
          {
          Blynk.setProperty(V5,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[4] > MCalib.StatusBorderPercentValues[4][0])
          {
          Blynk.setProperty(V5,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V5,"color",BLYNK_RED);
          }
      delay(100);
      Blynk.virtualWrite(V5,MMeasure.Percent[4]);
      } else
      {
      if (Calibrated)
        {
        Blynk.setProperty(V5,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V5,0);
        delay(100);
        Blynk.setProperty(V5,"color",BLYNK_BLACK);
        }
      else
        {
        Blynk.virtualWrite(V5,0);
        delay(100);
        Blynk.setProperty(V5,"color",BLYNK_BLUE);
```

```
        }
       }
     break;
     }
    case 5:
     {
     if ((MMeasure.DataValid[5]) & (Calibrated))
       {
       if ( MMeasure.Percent[5] > MCalib.StatusBorderPercentValues[5][1])
          {
          Blynk.setProperty(V6,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[5] > MCalib.StatusBorderPercentValues[5][0])
          {
          Blynk.setProperty(V6,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V6,"color",BLYNK_RED);
          }
       delay(100);
       Blynk.virtualWrite(V6,MMeasure.Percent[5]);
       } else
       {
       if (Calibrated)
         {
         Blynk.setProperty(V6,"label","Deaktiviert");
         delay(100);
         Blynk.virtualWrite(V6,0);
         delay(100);
         Blynk.setProperty(V6,"color",BLYNK_BLACK);
         }
       else
         {
         Blynk.virtualWrite(V6,0);
         delay(100);
         Blynk.setProperty(V6,"color",BLYNK_BLUE);
         }
       }
     break;
     }
    case 6:
     {
     if (DHTMeasure.DataValid)
       {
       Blynk.virtualWrite(V7,DHTMeasure.Humidity);
       } else
       {
       Blynk.setProperty(V7,"label","Deaktiviert");
       delay(100);
       Blynk.virtualWrite(V7,0);
       delay(100);
       Blynk.setProperty(V7,"color",BLYNK_BLACK);
       }
     break;
     }
    case 7:
     {
     if (DHTMeasure.DataValid)
       {
       Blynk.virtualWrite(V8,DHTMeasure.Temperature);
       } else
       {
       Blynk.setProperty(V8,"label","Deaktiviert");
       delay(100);
       Blynk.virtualWrite(V8,0);
       delay(100);
       Blynk.setProperty(V8,"color",BLYNK_BLACK);
       }
     break;
     }


    } // End Switch
  }

void Get_Moisture_DatainPercent()
 {
 byte CalibDataOffset = 0;
```

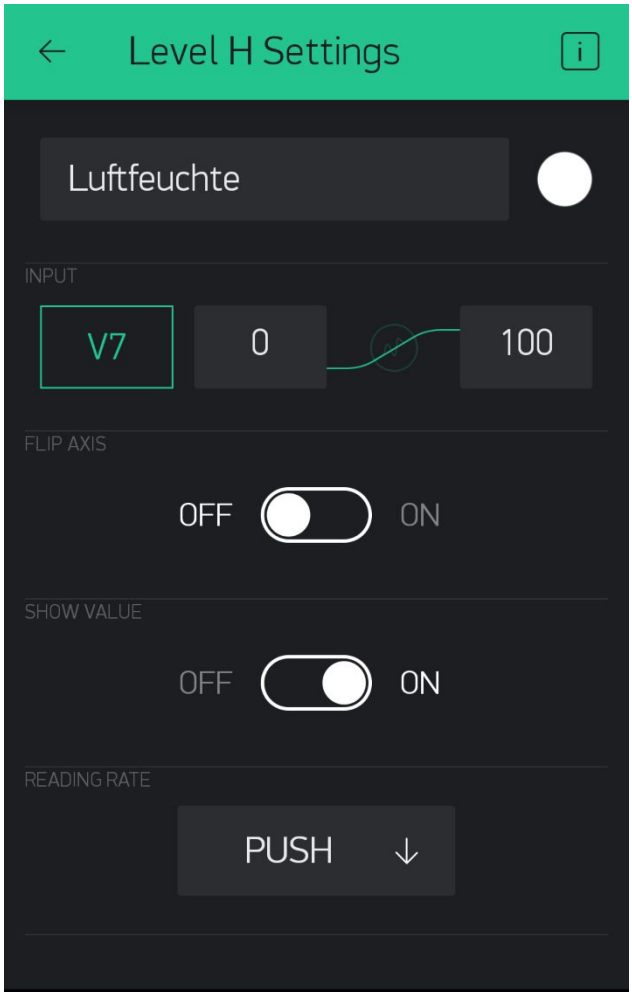
```
 for (byte i = 0;i < AttachedMoistureSensors;i++)
 {
 CalibDataOffset =  i *2;
 int RawMoistureValue = ReadMoistureSensor_Raw_Val(i);
 if ((MCalib.Data[CalibDataOffset] == 0) || (MCalib.Data[CalibDataOffset+1] == 0)) // MinADC Value maxADC ADC Value
   {
   MMeasure.Percent[i] = RawMoistureValue;
   MMeasure.DataValid[i] = false;
   } else
   {
   RawMoistureValue= MCalib.Data[CalibDataOffset+1] - RawMoistureValue;
   RawMoistureValue=MCalib.Data[CalibDataOffset] + RawMoistureValue;
   MMeasure.Percent[i] = map(RawMoistureValue,MCalib.Data[CalibDataOffset],MCalib.Data[CalibDataOffset+1], 0, 100);
   if ((MMeasure.Percent[i] > 100 ) | (MMeasure.Percent[i] < 0 ))
     {
     MMeasure.Percent[i] = RawMoistureValue;
     MMeasure.DataValid[i] = false;
     } else  {  MMeasure.DataValid[i] = true; }
   }
 }
return ;
}



void Run_MoistureSensors (bool Init)   // HauptFunktion zum Betrieb der Bodenfeuchtesensoren
{
byte MinSensValue = 100;
if ((millis() - Moisure_ServiceCall_Handler >= MoisureSens_Poll_Interval) | (Init))
 {
  Moisure_ServiceCall_Handler = millis();
  Get_Moisture_DatainPercent();
  for (int i = 0;i < AttachedMoistureSensors;i++)
    {
      if (MMeasure.DataValid[i])
        {
        if (MMeasure.Percent[i] != MMeasure.Old_Percent[i])
          {
          MMeasure.Old_Percent[i] = MMeasure.Percent[i];
          if (MMeasure.Percent[i] < MinSensValue ) { MinSensValue = MMeasure.Percent[i]; };
          Serial.print(F("Feuchtigkeitswert Sensor "));
          Serial.print(i);
          Serial.print(F(" in Prozent :"));
          Serial.print(MMeasure.Percent[i]);
          Serial.println(F(" %"));
          Update_Blynk_APP(i,Sens_Calib);     // Aktualisiere Handywerte
          }
        } else
        {
         Update_Blynk_APP(i,Sens_NOTCalib);     // Aktualisiere Handywerte
         Serial.print(F("Sensor "));
         Serial.print(i);
         Serial.print(F(" nicht kalibiert. Bitte kalibrieren. Rohdatenwert:"));
         Serial.println(MMeasure.Percent[i]);
        }
    }
  Update_Local_Display();          // Aktualisiere lokales Pflanzenwächter Display (Led)
 }
}

bool Run_DHTSensor (bool Init)   //
{
if ((millis() - DHT_ServiceCall_Handler >= DHT_Poll_Interval) | (Init))
 {
 DHT_ServiceCall_Handler = millis();
 DHTMeasure.Humidity = dht.readHumidity();
 DHTMeasure.Temperature = dht.readTemperature(false);   // Read temperature as Celsius (isFahrenheit = true)
 if (isnan(DHTMeasure.Humidity) || isnan(DHTMeasure.Temperature) )
   {
   Serial.println(F("Failed to read from DHT sensor!"));
   DHTMeasure.DataValid  = false;
   return false;
   }
 DHTMeasure.DataValid  = true;
 if (DHTMeasure.Humidity != DHTMeasure.Old_Humidity)
    {
    DHTMeasure.Old_Humidity = DHTMeasure.Humidity;
```

```
    Update_Blynk_APP(6,true);  //  Luftfeuchteanzeige
    }
  if (DHTMeasure.Temperature !=  DHTMeasure.Old_Temperature)
    {
    DHTMeasure.Old_Temperature = DHTMeasure.Temperature;
    Update_Blynk_APP(7,true);  //  Temperaturanzeige
    }

  }
return true;
}


// Main Loop
void loop()
{
 Run_MoistureSensors(RunTime);
 if (DHTMeasure.SensorEnabled) { Run_DHTSensor(RunTime); }
 Blynk.run();   // Execute Blync Basic- Functions
}
```

Weitere unbedingt anzupassende Parameter sind nicht hinzugekommen, da der DHT 22 und auch der DHT 11 Sensor eine interne Werte Kalibrierung hat, die keiner weiteren Anpassung mehr Bedarf.

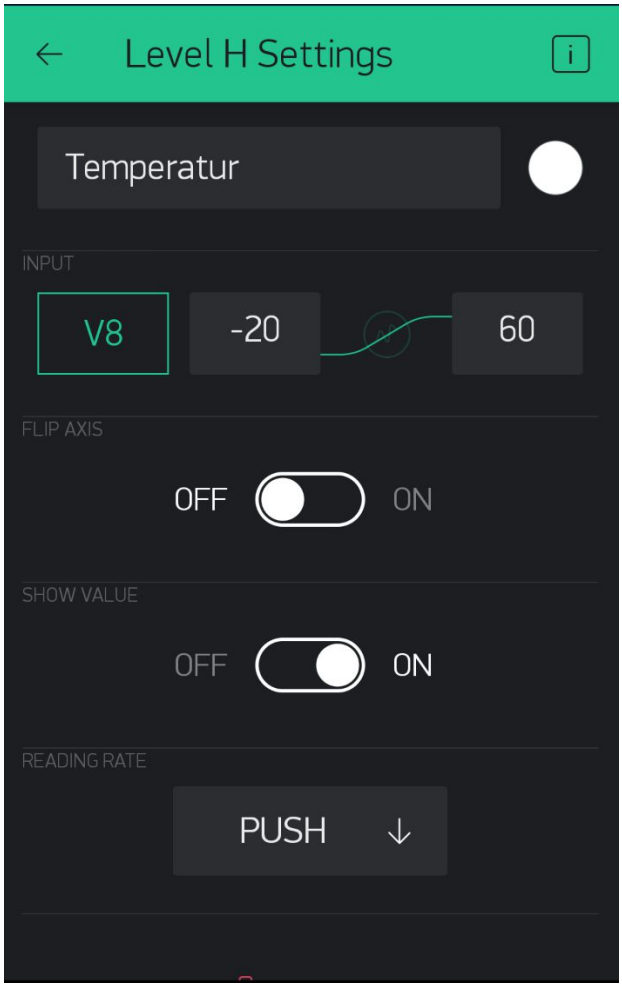
Nachdem wir den Code nun angepasst und auf unseren ESP hochgeladen haben und sich unser ESP erfolgreich in das WLAN eingeloggt hat (Weiße LED geht aus) müssen wir auch in diesem Teil wieder unsere APP erweitern.

Dazu fügen wir in unsere APP zwei weitere „LEVEL H“ Elemente hinzu und konfigurieren das erste Element für die Luftfeuchte wie folgt:



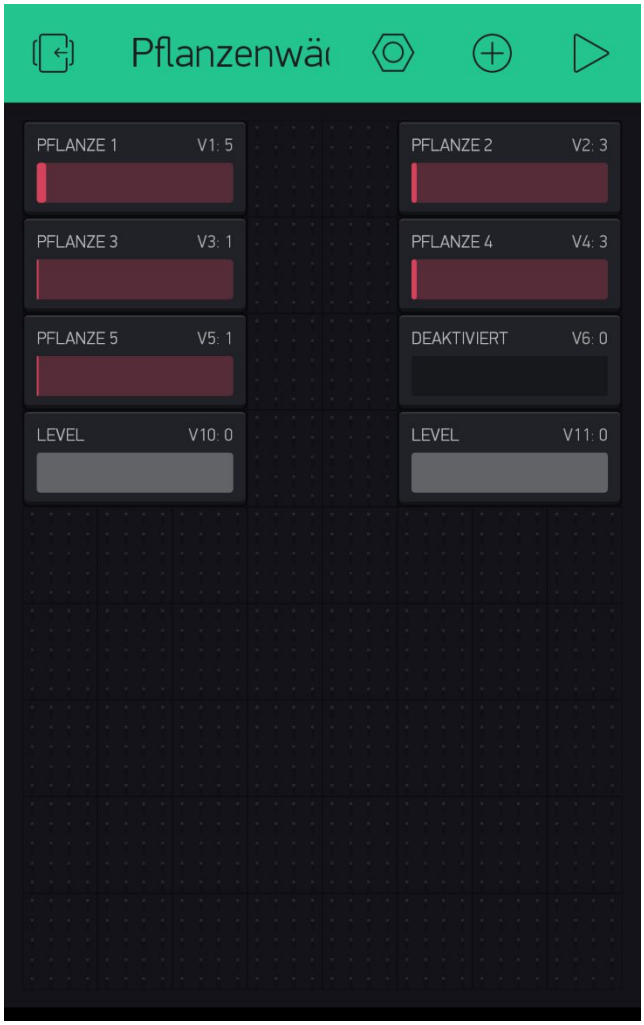
- Farbe: Weiß
- Input Pin: V7 (virtuell)
- Minimum Wert:0
- Maximum Wert: 100
- Reading Rate: PUSH

Das zweite „Level H“ Element für die Temperatur konfigurieren wie folgt:



- Farbe: Weiß
- Input Pin: V8 (virtuell)
- Minimum Wert: -20
- Maximum Wert: 60
- Reading Rate: PUSH

Anschließend sollte unsere Pflanzen App wie folgt aussehen:



Nach drücken des „Start“ Buttons bekommen wir nun in der APP neben den aktuellen Bodenfeuchtesensorwerten auch die aktuelle Temperatur und Luftfeuchtigkeit dank unserem DHT Sensor angezeigt.

Ich wünsche viel Spaß beim Nachbauen, und bis zum nächsten Mal.